A.1 basic_feature_extractor
here we just took the test images and converted them to 1's and 0's thereby removing the "edge" feature

A.2

If we are given a large number number of features and used standard multiplicative representation of joint probability we will be multiplying a lot of small decimal numbers with each other. Eventually you will get a number extremely small, so small the computer may not be able to represent the number as a decimal and may round it to 0.

A.3 here we computed the prior probability and the conditional probability from all the test images. For each test image we used the label to identify it, count its frequency among st test images, and divide it by total test images to get prior probability. For conditional probability we looped through all the images and found the total amount of times an image was foreground for a certain row/column pixel for each number. Then we divided each of those totals for each number by the numbers frequency to get the probability a pixel is foreground for each row/column pixel. This was our conditional probability.

A.4 in classify and in compute class we used the computed statistics to determine the bayes rule probabilities a test image was certain number for all possible numbers. Then we take the max of those probabilities and make that our predicted number.

A.5

Percentage	accuracy
10%	.579
20%	.586
30%	.584
40%	.596
50%	.608
60%	.62
70%	.618
80%	.621
90%	.623
100%	.63

Here we can see that as our program is fed more test images we in general get a better accuracy reading. In some cases the accuracy goes down a little. This may be because we feed our algorithm bad or outlier sample images to learn from that skews our probabilities.

A.6
After testing our code with multiple K values we noticed that the larger the k was the worst the accuracy became. Thus the smaller the K was the best accuracy would result. We thus chose the

smallest integer k value, that being 1. We tested even further than that with decimal k's and noticed a .01 increase in accuracy with .000001 k. after adding more 0's at that point there was very marginal increase in accuracy.

B.1
We decided to use 3 new features that we believed would improve the accuracy of our code. As seen in the above chart we were able to achieve 63 percent accuracy from our extract_basic_features. In extract basic features we use the test data from the 5000 test images and came up with the probability a pixel was foreground given a certain number. Having this conditional probability at our disposal, we would take in a test image, find where all the pixel were foreground in the image and then use bayes rule with our conditional probability to decide which of the 10 digits we thought was most likely.

The three new features we implemented were:
1) use probability a certain pixel is background for certain numbers
2) probability a certain pixel is an edge
3) smooth out the picture

Rationale:

Previously we only took the foreground of a number and use that to calculate the likeliness a test image was a certain number but this was limiting the search space and essentially ignoring the background. However, now we expand the area to include the background probability as well thus improving our prediction immensely. We can do this by taking in a test image and traversing every pixel. If we see a background pixel, we add (1 – probability(foreground)) to our log Bayes rule formula. If advance is called, we only take the probability a pixel is background. Using the same thought process from extract_basic_features for foreground, we take the max probability of the 9 numbers and make that our predicted number.

We also gave more probability to edges since it was more clearly defined. This favored the edges more and took edges more into account than the inner pixels. If an edge was more likely to be in an area of the image we would use the same logic as the foreground and background. From there we sse bayes rule to calculate the likeliness an image was certain number based on where the edges were on the test image and take the max probability of the 9 digits.

Lastly, we decided to smooth the sample data out. When we call extract_advance_features what we did was we saw that there were sometimes holes or inconsistencies in many of the test image foregrounds. So, do better improve the sample data we would fill those holes and "smooth" out the images. Essentially if a pixel was surrounded by three or more pixels that were in the foreground, we set that pixel to true. We believe that by doing this, we are reducing the discrepancies of the different images to better uniform those images. If we did this over all 5000 images, we believed that we would marginally increase the accuracy of our predictions.

Extract_advanced_features

PERCENTAGE OF TEST DATA	ACCURACY
10%	.688

20%	.693
30%	.703
40%	.721
50%	.719
60%	.73
70%	.722
80%	.735
90%	.74
100%	.742

Again we see here that a higher amount of images leads to more accurate data. We see a difference of about 6% when we go from 500 images to 5000 images.

B.2 Performance with both basic and advance

10%	.75
20%	.779
30%	.789
40%	.797
50%	.805
60%	.797
70%	.806
80%	.815
90%	.813
100%	.815

Just like before, we see an increase in accuracy when going from 500 images to 5000 images. It should be noted that even testing on just 500 images, our results are still better than when extract_advanced_features was tested on 5000 images. This shows how much the basic feature adds to our accuracy.

B.3 refer to code we implemented both basic and advance and got the best results as seen in part B.2